Squirrels & Wolves

Computação Paralela e Distríbuida - MEIC

Nelson Silva, Rafael Baltazar, Samuel Coelho

Instituto Superior Técnico - Alameda

1. Estrutura da solução

Para garantir a execução correta dos conflitos, decidimos utilizar duas matrizes.
No início de cada subgeração, estas duas matrizes são iguais, mas uma delas é declarada como *read* e outra como *write*. Têm este nome, porque não deverá existir nenhuma operação de escrita na matriz *read* e apenas se lerá da matriz *write* em raras situações.
Desta forma, a cada sub-geração, são processados todos os animais da matriz *read* e todos eles irão ver os seus vizinhos na matriz *read* e mover-se para a matriz *write*. Ao mover-se, é possível gerar um conflito e será essa a única vez, durante uma sub-geração, que será preciso ler da matriz *write*.
É também, quando se lê da matriz *write* para depois escrever nela, a única situação em que poderá ocorrer *data racing*. Resolvemos o problema protegendo essa parte do código através de #pragma omp critical(move).

2. Distribução do trabalho pelas threads

A versão paralela utiliza paralelismo em três situações: a cada sub-geração, e no final de cada geração.
Para cada sub-geração, o processamento das linhas da matriz é distribuído pelas *threads*, através da directiva #pragma omp parallel for, ou seja, cada *thread* processa as células pertencentes às linhas de que é responsável.
No final de cada geração, é necessário actualizar os períodos de *breeding* e *starvation* de cada animal através da função *update_periods*. Como apenas é necessário ler e escrever na mesma célula, o processamento das linhas é, mais uma vez, igualmente distribuído pelas *threads*.

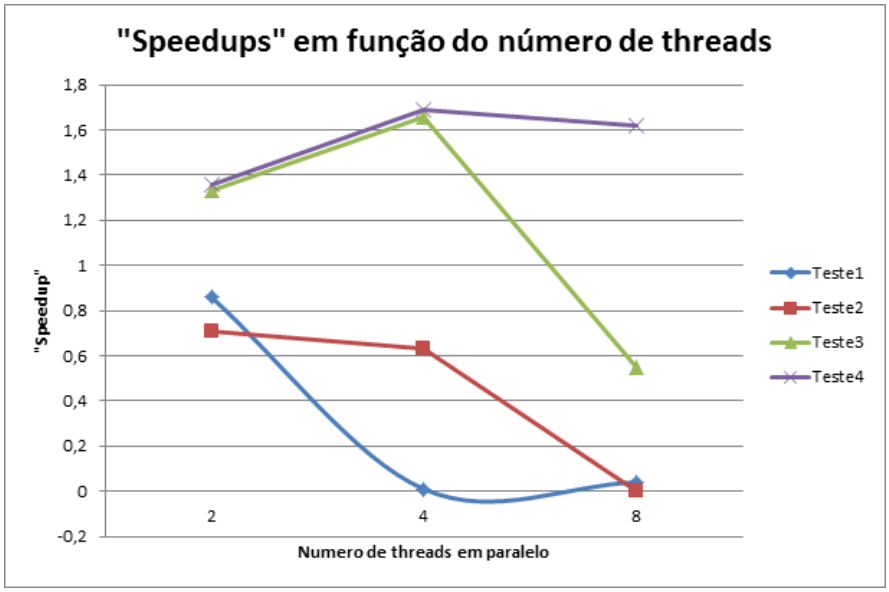
3. Speed-up

Para testar o projecto foram executados um conjunto de testes cuja descrição se encontra na tabela abaixo. A máquina utilizada para a execução dos testes possuía um processador com 2 cores e hyper-threading, isto é, cada core pode executar 2 threads ao mesmo tempo.
Os ficheiros de input referidos na tabela correspondem aos fornecidos pelo corpo docente da disciplina.

Nome	Ficheiro de teste	wolf breeding	squirrel breeding	wolf starvation	# gerações
T1	world_10.in	20	5	30	5
T2	world_10.in	20	50	30	1000 000
T3	world_100.in	20	50	30	100 000
T4	world_1000.in	20	50	30	10 000

A tabela seguinte descreve os resultados da execução dos testes.

Teste	N.º de threads	tsequencial (s)	tparalelo (s)	Speed-up
T1	2	0.006	0.007	0.86
T1	4	0.006	0.981	0.01
T1	8	0.005	0.012	0.04
T2	2	4.587	6.43	0.71
T2	4	4.627	7.35	0.63
T2	8	4.588	18m17.1	0
T3	2	1m17.373	58.275	1.33
T3	4	1m18.566	47.234	1.66
T3	8	1m17.301	2m19.589	0,55
T4	2	7m28.753	5m30.726	1.36
T4	4	7m41.832	4m32.559	1.69
T4	8	7m28.753	4m36.386	1.62



Conclusão:

Quanto maior o tamanho da matriz, maior será o trabalho realizado por cada thread, pois cada linha terá mais células e cada thread processará mais linhas. Consequentemente, o speed-up aumentará.

No entanto, o valor do speed-up tende a diminuir quando são criadas mais threads do que o número de processadores da máquina. Como referido anteriormente, a máquina utilizada tinha um processador que, na prática, pode ser considerado um quad-core. Como é possível observar pelo gráfico, o speed-up diminui quando o número de threads é aumentado para 8.

Quanto mais gerações, menor o speed-up, pois são criadas e destruídas threads a cada sub-geração.

O speed-up não é apenas influenciado pelo número de threads pois o hardware limita bastante. Como tal, só vale a pena investir em paralelismo se existir muito trabalho computacional por cada tarefa que é executada em paralelo e se estivermos na posse do hardware adequado ao efeito.